(2)

Naval Research Laboratory

Washington, DC 20375-5000

NRL Report 9282

Enhancing Multidimensional Tree Struc ures by Using A Bi-Linear Decomposition

JEFFREY K. UHLMANN

Battle Management Technology Branch
Information Technology Division

November 19, 1990

REPORT DOCUMENTATION PAGE	Form Approved OMB No 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE November 19, 1990	3. REPORT TYPE AND DATES COVERED

4. TITLE AND SUBTITLE Enhancing Multidimensional Tree Structures by Using a Bi-Linear Decomposition	5. FUNDING NUMBERS PE - 63223C PN - 55-2354-N0 WU - DN155-097
6. AUTHOR(S) Jeffrey K. Uhlmann	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Research Laboratory Washington, DC 20375-5000	8. PERFORMING ORGANIZATION REPORT NUMBER NRL Report 9282
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Strategic Defense Initiative Organization 1717 K Street, NW Washington, DC 20006	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

A performance of *k*-d trees for identifying near-neighbors can be improved through the use of a bi-linear decomposition. This gain in performance is obtained by enhancing the tree structure so that it can not only store where things are, but also where they are not. This approach is particularly advantageous when the search and query sets are correlated.

14. SUBJECT TERMS Bi-Linear decomposition Multidimensional search *k*-d trees	Range searching Computational geometry Correlation	15. NUMBER OF PAGES 12 16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

NSN 7540-01-280-5500

Standard Form 298 (Rev 2-89)
Prescribed by ANSI Std Z39-18
298-102

CONTENTS

ENHANCING MULTIDIMENSIONAL TREE STRUCTURES BY USING A BI-LINEAR DECOMPOSITION

INTRODUCTION

The k-d tree data structure ~~[1]~~ provides a relatively distribution-independent means for satisfying orthogonal range queries on k-dimensional objects in average case time. This time is proportional to the theoretic optimum for any data structure whose storage requirements scale linearly ~~[2]~~. Given this optimality, much work in the area of distribution-independent near-neighbor algorithms has been directed toward enhancing the k-d tree and its associated search techniques to reduce its search time proportionality constant. This report suggests an enhancement that furthers this end. Important applications include data correlation problems associated with multitarget tracking. In particular, techniques described in this report have been incorporated into the TRC ~~[3,4,5]~~ and REAL ~~[6,7]~~ tracking and correlation systems.

A k-d tree is a binary tree in which the set of k-dimensional points may be partitioned at each node according to any of the k coordinates. The discriminating coordinate for each node can be selected to prevent anisotropy in the distribution ~~[8]~~. This is accomplished by recursively partitioning the set according to the median data point of the projection of the data onto the coordinate axis that has the greatest dispersion. Thus, a node will contain (either implicitly or explicitly) identification of the discriminating coordinate and pointers to the set of points whose values for the discriminating coordinate are greater than that of the median and to the set of points whose values are less than the median. The methods for searching the tree are then completely analogous to those used to search ordinary single-dimension binary trees. For range searching, the worst-case scaling for a single query is $O(N^{1-\frac{1}{k}} + m)$, where N is the size of the dataset, k is the number of dimensions, and m is the number of neighbors returned. In many applications, however, the average-case scaling approaches $O(k \log^k N + m)$.

THE HIERARCHICAL DIVISIBILITY PROBLEM

The problem with the k-d tree is what has been called the hierarchical divisibility (HD) problem [9]. This problem results from the manner in which a tree structure recursively partitions a spatial region into smaller subregions. Each node in a tree represents a hyperplane that divides the space and thus partitions the dataset. However, if a near-neighbor query defines a neighborhood that is intersected by one of these hyperplanes, then at least two paths beginning at the node associated with the plane must be examined to the full depth of the tree to determine if any more points are in the neighborhood. Even if no plane intersects the neighborhood, a complete path from the root to a terminal node must still be examined. In short, the HD problem arises because a finite interval can be infinitely subdivided.

Manuscript approved August 8, 1990.

As an example, consider a trivial one-dimensional case in which a query is made to find all the data points on the interval [50, 60] in the set {49, 97, 7, 91, 13, 45, 85. 19, 61, 55, 73, 31, 67, 25, 37, 79}. Figure 1 shows its balanced tree (whose nonterminal nodes are computed from the average of the high and low values of their subtrees).

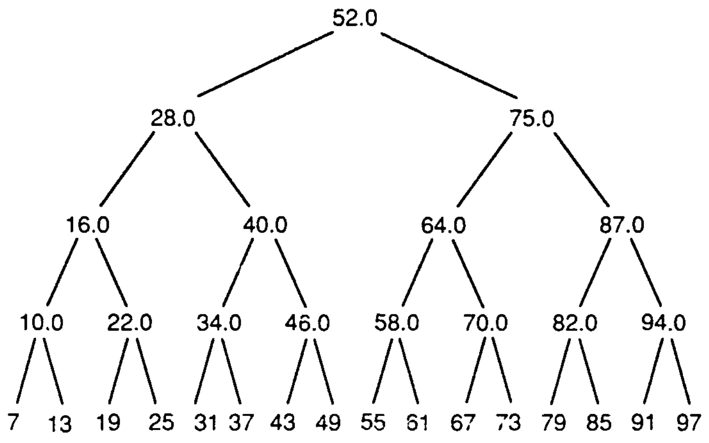

Fig. 1 — Balanced tree

After examining the number 52.0 at the root, the search must examine the set of values that are less than 52.0 to determine whether its largest elements fall within the range. The set of values larger than 52.0 must then also be examined to determine whether its smallest elements fall within the range. Because the extreme values in a balanced tree are always associated with terminal nodes, the subtrees associated with these two sets must be traversed from their roots to at least one terminal node (Fig. 2).

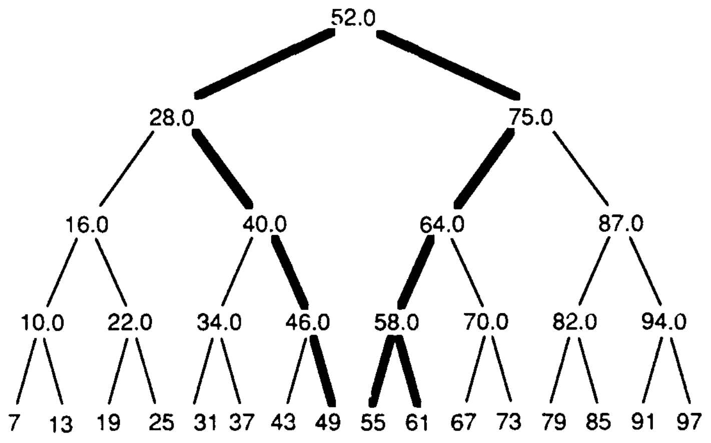

Fig. 2 — Search path through the *k*-d tree

This problem is exacerbated by a correlation between the query intervals and the search set because the partitioning planes are also correlated with the search set. In other words, such a correlation increases the likelihood at some level in the tree that each query interval will straddle a partitioning plane.

THE BI-LINEAR DECOMPOSITION

The solution to the HD problem is to construct a bi-linear decomposition (BLD). A BLD differs from an ordinary binary tree in that each of its nodes represents two partitioning hyperplanes rather than one. This pair of planes delimits a k-dimensional volume that can be eliminated from the search space. The partitioning planes are determined simply by identifying the *two* median data points in the coordinate projection at the current level of the tree. The following pseudocode demonstrates the simplicity of the construction process:

```
< This recursive function accepts as parameters a set S of
  data points, the FIRST index into the array, the LAST
  index into the array, and the coordinate D according to
  which the set is to be partitioned. >

function BLD (S, FIRST, LAST, D ):

    < The following four lines associate a point with a
      nonterminal node and then returns. >

    if FIRST equals LAST then

        NODE.POINT_INDEX = FIRST;

        return(NODE);

    endif;

< The following line sorts the array of data points by
  the Dth coordinate into ascending order. Actually,
  only the middle two elements of the sorted array need
  to be in their proper positions; hence, an 0(n)
  routine [10] may be substituted for the 0(nlogn)
  sort. This substitution is necessary to ensure that
  the construction process requires only 0(nlogn) time. >

sort (S, FIRST, LAST, D);

< The following two lines associate two hyperplanes
  (which delimit the empty space in the Dth dimension
  between the data points in the two sets) with the
  nonterminal node. >

NODE.LEFT_HYPERPLANE = S[(FIRST+LAST)/2].DIMENSION[D];

NODE.RIGHT_HYPERPLANE = S[((FIRST+LAST)/2)+1].DIMENSION[D];

< The following four lines determine the new discrimi-
  nating coordinate and then invoke BLD on the two
  sets and assigns the results to the node's left and
  right pointers. >
```

```
        if D equals the number of dimensions then D = 1;
        else D = D + 1;
        NODE.LEFT = BLD(S, FIRST, (FIRST+LAST)/2, D);
        NODE.RIGHT = BLD(S, NODE.LEFT+1, LAST, D);
        return(NODE);

    end.
```

Using this procedure on the data from the previous example, the tree shown in Fig 3 is produced.

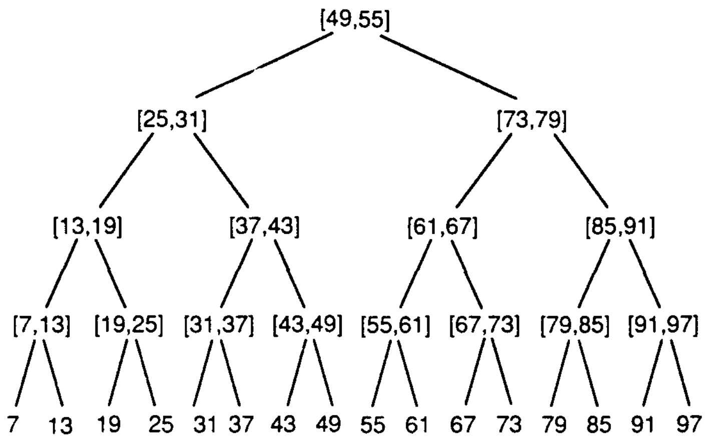

Fig. 3 — BLD form of one-dimensional, balanced tree

The philosophy of the BLD is that it is more efficient to determine the location of something by first determining where it is *not* located; hence, nonterminal nodes are used to identify hyperplanes that delimit empty space. Thus, to find points on the interval [50, 60], compare the value 50 with one hyperplane while comparing the value 60 with the other. If 50 is less than or equal to the value representing the 'left' hyperplane, then the search must include the set of nodes pointed to by the node's left pointer. Similarly, if 60 is greater than or equal to the value representing the 'right' hyperplane, then the search must include the set of nodes pointed to by the node's right pointer (Fig. 4).

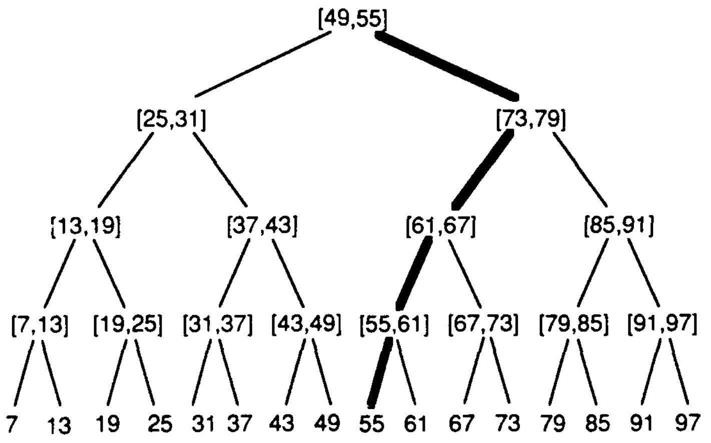

Fig. 4 — Search path through the BLD tree

In this example, therefore, only five nodes are visited (as opposed to the 10 nodes that must be visited when using the traditional tree approach).

EXTENSIONS TO THE VOLUMETRIC CASE

An important related class of problems requires the efficient identification of intersecting volumes. For practical reasons, it is often assumed that the volumes are approximated by isothetic rectangles, i.e., boxes whose sides are parallel to the coordinate axes. However, since the projection of boxes onto an axis results in a set of intervals, possibly overlapping, the notion of a projection median that can always divide a dataset into two disjoint sets can no longer be applied. Thus, a multidimensional tree that is enhanced to handle volume queries cannot be strictly binary.

The multidimensional ternary tree search structure is a ternary tree in which the set of d-dimensional boxes is partitioned at each node according to one of the d coordinates. The resulting tree is ternary because at each node the set is partitioned into a set of boxes that lie entirely to the left of the partitioning plane, a set of boxes that lie entirely to the right of the partitioning plane, and a set of boxes that are intersected by the partitioning plane. The method for searching the tree then simply involves the determination of which set the search box would be assigned according to the partitioning plane at each node. For example, if the search box lies entirely to the left of the partitioning plane, then the subtree of boxes that lie entirely to the right of it can be ignored. Similarly, if the search box lies entirely to the right of the partitioning plane, then the subtree of boxes that lie entirely to the left of it can be ignored. In the case where the partitioning plane intersects the search box, all subtrees must be examined.

As in the k-d tree case, the levels at which the query range intersects the partitioning plane contribute significantly to the overall cost of the search. This is because at least three paths beginning at the node associated with the plane must be examined to the full depth of the tree. In direct analogy to the binary tree, a BLD enhancement can be made to the ternary tree structure. In the ternary case, however, the pair of planes delimits a d-dimensional volume for which only the middle subtree must be examined. The partitioning planes can be identified by finding the median of the left endpoints of the projection intervals and partitioning the dataset into three sets, as described above. The left and right partitions are then simply the max and min values, respectively, of the left and right sets of intervals.

TEST RESULTS

The following test results (performed on a Sun-260) are provided to compare a leaf-oriented, balanced k-d tree to a BLD-enhanced version. These results demonstrate the effect of variables such as dataset size, neighborhood size, and number of dimensions on the relative performances of the two algorithms for uniformly distributed point sets. Measurements are made of the setup time for the data structure, the search time required for n queries (where n is the number of data points in the system), the average number of neighbors found per query, and the average number of distance calculations required per query. In the construction of both data structures, the discriminating coordinates are chosen in a nonpreferential, cyclical fashion to avoid subtle selection-dependent effects on relative performance. Even for uniformly distributed points, however, such a selection strategy significantly degrades performance as the number of dimensions increases.

TEST 1

In test 1, the algorithms are run on 8 K, 16 K, and 32 K datasets in three dimensions with neighborhoods that have an average of five points. Table 1 shows that the BLD is approximately 15% faster than the *k*-d tree and that this increase is independent of the size of the dataset.

Table 1 — Test 1 Results

	Test 1.1		Test 1.2		Test 1.3	
	BLD	Tree	BLD	Tree	BLD	Tree
Number of objects (*N*)	8192	8192	16384	16384	32768	32768
Number of dimensions	3	3	3	3	3	3
Setup time (s)	2.0	2.0	4.3	4.3	9.4	9.4
Search time (*N* queries)	10.6	12.6	22.5	26.5	44.8	53.0
Avg. no. of neighbors	5	5	5	5	5	5
Avg. no. of nodes visited	51.9	56.3	54.7	59.0	57.0	61.4
Avg. no of dist. calcs.	20.4	26.4	20.7	26.7	19.7	26.0

TEST 2

In test 2, the algorithms are run on a 16 K dataset in three dimensions with neighborhoods that have averages of 5, 10, and 20 points. Table 2 shows that the advantage in speed decreases slightly as the size of the neighborhood increases. More extensive tests show that the BLD maintains an advantage for small (relative to the size of the system) neighborhoods. Clearly, if the neighborhood includes every point in the system, searching each structure (i.e., BLD and tree) requires visiting every node.

Table 2 — Test 2 Results

	Test 2.1		Test 2.2		Test 2.3	
	BLD	Tree	BLD	Tree	BLD	Tree
Number of objects (*N*)	16384	16384	16384	16384	16384	16384
Number of dimensions	3	3	3	3	3	3
Setup time (s)	4.3	4.3	4.3	4.3	4.3	4.3
Search time (*N* queries)	22.5	26.5	37.1	43.1	60.9	68.8
Avg. no. of neighbors	5	5	10	10	20	20
Avg. no. of nodes visited	54.7	59.0	80.5	86.4	121.7	129.7
Avg. no of dist. calcs.	20.7	26.7	37.0	45.5	65.5	77.6

TEST 3

In test 3, the algorithms are run on a 16 K dataset in 2, 4, and 8 dimensions with neighborhoods that have an average of five points. Table 3 shows the BLD to have an execution speed advantage of approximately 15% independent of the number of dimensions. (As mentioned earlier, a more sophisticated strategy for selecting the discriminating coordinates when constructing the data structures can dramatically improve performance as the number of dimensions increases. Intuitively, this is because the number of points required to approximate a uniform distribution increases exponentially with the number of dimensions.)

Table 3 — Test 3 Results

	Test 3.1		Test 3.2		Test 3.3	
	BLD	Tree	BLD	Tree	BLD	Tree
Number of objects (*N*)	16384	16384	16384	16384	16384	16384
Number of dimensions	2	2	4	4	8	8
Setup time (s)	4.3	4.3	4.3	4.3	4.4	4.4
Search time (*N* queries)	9.4	10.8	54.6	66.2	1677.2	1984.7
Avg. no. of neighbors	5	5	5	5	5	5
Avg. no. of nodes visited	28.7	30.8	111.2	120.5	1743.0	1856.3
Avg. no of dist. calcs.	8.9	11.6	48.7	62.2	1038.0	1271.6

SUMMARY

This report describes an enhancement to the standard multidimensional tree data structure that modestly improves the execution speed for the satisfaction of orthogonal range queries. This enhancement, called a bi-linear decomposition (BLD), is straightforward to implement and can be adapted for volumetric intersection queries. Results of tests of the BLD approach reveal that a reduction in the number of nodes visited and in the number of distance calculations required yield the speed improvements. Minimizing node visitations is important in distributed processing environments where nodes are mapped to processors and where the cost of interprocessor communication is significant.

REFERENCES

1. J.L. Bentley, "Multidimensional Binary Trees for Associative Searching," *Commun. ACM* **18**(9), 509-517 (1975).

2. K. Mehlhorn, *Multidimensional Searching and Computational Geometry*, (Springer-Verlag, Berlin, 1984).

3. Ball Systems Engineering Division, "SDI Midcourse Tracker/Correlator Near Neighbor Algorithm Integration and Testing," BSED report to NRL, Jan. 25, 1990.

4. J. Uhlmann, M. Zuniga, and J.M. Picone, "Efficient Approaches for Report/Cluster Correlation in Multitarget Tracking Systems," NRL Report 9281, 1990.

5. M. Zuniga, J.M. Picone, and J. Uhlmann, "An Efficient Algorithm for Improved Gating Combinatorics in Multiple-Target Tracking," submitted to *IEEE Trans. Aerospace Electron. Systems*, April 1990.

6. J. Collins and J. Uhlmann, "REAL Approach to Tracking and Correlation for Large-Scale Scenarios," *NRL Review*, 1989-1990.

7. J.B. Collins and J.K. Uhlmann, "Efficient Data Association for Multivariate Gaussian Distributions," submitted to *IEEE Trans. Aerospace Electron. Systems*, Feb. 1990.

8. J.A. Friedman, J.L. Bentley, and R.A. Finkel, "An Algorithm for Finding Best Matches in Logarithmic Expected Time," *ACM Trans. Mathe. Software*, **3**(3), 209-226 (1977)

9. J. Uhlmann, "A Remedy for the Hierarchical Divisibility Problem Using a Bi-Linear Decomposition," Berkeley Research Technical Report BRA-89-W040R (1988).

10. D. Knuth, *Sorting and Searching*, (Addison-Wesley, Reading, MA, 1973).